Component Infrastructure for Managing Performance Data and Runtime Adaptation of Parallel Applications

Li Li,[1] Boyana Norris,[1] Henrik Johansson,[2] Lois Curfman McInnes,[1] and Jaideep Ray[3]

[1] Argonne National Laboratory, Argonne, IL, USA,
[likli,norris,mcinnes]@mcs.anl.gov
[2] Department of Information Technology, Uppsala University, Sweden
henrik.johansson@it.uu.se
[3] Sandia National Laboratories, Livermore, CA, USA,
jairay@sandia.gov

Abstract. Component-based software engineering (CBSE) has been gaining popularity in parallel scientific computing, facilitating the creation and management of large multidisciplinary, multideveloper application codes and providing opportunities for improved performance and numerical accuracy. The CBSE approach enables automation of traditionally manual application configuration and tuning tasks. In addition to encapsulating data and functions, components enable the encapsulation of nonfunctional properties (metadata) whose values affect the performance of a component. By first automatically collecting performance data and associated metadata, and then analyzing it to classify different methods, we can automate the selection of the best-performing component instance among many functionally equivalent implementations. To support this adaptivity, we have designed a set of interfaces and components for managing databases for performance and system information, analysis, and runtime control. We describe our initial implementations of these interfaces and discuss their use in selecting linear solvers in inexact Newton methods for nonlinear partial differential equations (PDEs) and dynamic selection of the most appropriate partitioning strategy at runtime in combustion applications.

1 Introduction

As computational science progresses toward ever more realistic multiphysics and multiscale applications, no single research group can effectively develop, select, or tune all of the components in a given application, and no single tool, solver, or solution strategy can seamlessly span the entire spectrum *efficiently*. Component-based software engineering approaches help manage some of the complexity of developing such large scientific applications. The Common Component Architecture (CCA) [3,6] defines a component software engineering approach specifically targeted at high-performance computing (HPC) applications. The CCA, like

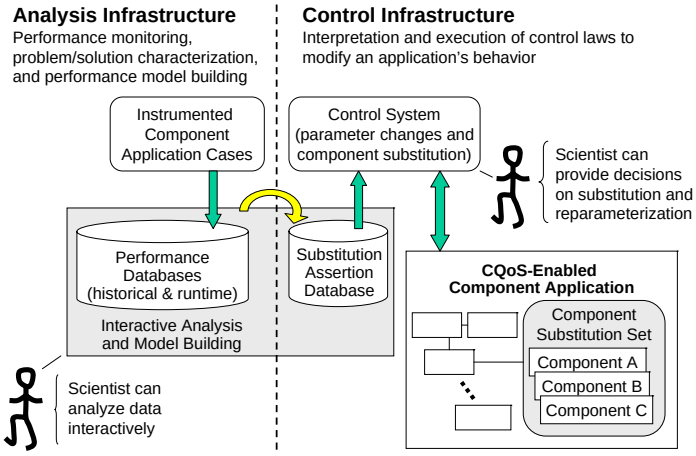

Fig. 1. CQoS middleware organization.

other component and service specifications, provides support only for basic manipulation of components, such as repositories, instantiation, connection, and execution. Common interfaces enable easy access to suites of independently developed algorithms and implementations, and dynamic composability facilitates switching among different implementations statically or during runtime. The challenge then becomes how to make sound choices from among the available implementations and parameters, with suitable tradeoffs among performance, accuracy, mathematical consistency, and reliability. Such choices are important both for the initial composition and configuration of an application and for adaptive control during runtime.

To at least partially automate the process of characterizing the performance of component applications and selecting and configuring particular implementations, we have introduced the concept of computational quality of service (CQoS) [20] for scientific applications. CQoS expands on traditional QoS ideas by considering additional application-specific metrics, referred to as metadata, that enable the characterization of the performance of high-performance components. Before automating the selection of component instances, however, one must be able to collect and analyze performance information and the related metadata.

The conceptual organization of the middleware necessary to support automated configuration and dynamic adaptation of component applications is illustrated in Figure 1.

In the remainder of this paper we focus on the performance database infrastructure portion of the CQoS architecture. We present CCA interfaces and

components that support the collection and management of performance data and associated metadata and their use in two different parallel application contexts.

2 Database Interfaces and Components

The database interface design is intended to support the management and analysis of performance and application metadata, so that the mapping of a problem to an implementation that can potentially yield the best performance can be accomplished statically or at runtime. Figure 2 shows the main interfaces and some of their methods.

We introduce two types of components for storing and querying CQoS performance data and metadata. The database component provides general-purpose interfaces for storing and accessing data in a physical database. The comparator interfaces compare and/or match properties of two problems under user-specified conditions.

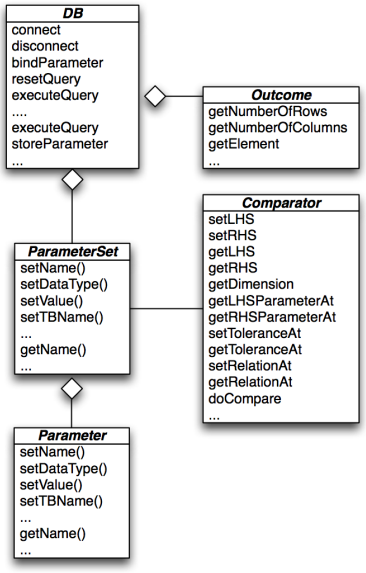

Fig. 2. Database interfaces.

2.1 Comparator Components

There are three sets of interfaces associated with a comparator component, *Parameter*, *ParameterSet*, and *Comparator*. A *Parameter* captures a single property of a problem, e.g., a scalar representing mesh quality. A parameter is described by its name, data type, and value. It is associated with a table in the physical database. The *Parameter* interfaces also support comparisons against another peer parameter under user-specified conditions. A *ParameterSet* represents a group of related parameters, e.g., a set of parameters that characterize an adaptive grid, or a set of scalar or boolean linear system properties. Using the *ParameterSet* interfaces, users can create and manage parameter set members. When selecting a solution method, the time-dependent problem/system properties are described as one or more *ParameterSets*. The user or an automated adaptive heuristic can then match the formatted parameter sets to a database to determine the best known solution method or configuration. A *Comparator* defines the rules to compare two sets of parameters. For instance, a *Comparator* can determine the closeness of two sets of parameters, i.e., whether they are within ϵ of each other.

2.2 Database Components

There are two classes of interfaces associated with a database component, *DB* and *Outcome*. The application connects to a database component by using the *DB* port, which handles (potentially remote) database connections, queries, and storage and retrieval of parameters and parameter sets. The *DB* interface also supports the query of experimental runs having parameter sets that satisfy user-specified conditions (e.g., limiting the parameter set to a range of values). The *Outcome* interface supports transformation of database results obtained by using a DB query to user-readable format, as well as access to the individual data elements.

The results of exhaustive analyses of properties for a number of problem instances can also be stored in the database using the database interfaces. Before the execution of an application, application-specific *Comparator* implementations help match the initial problem properties and system states against historical information to find a good initial solution method. During runtime, time-dependent application and system characteristics are captured in metadata parameter sets. A runtime *Comparator* implementation can dynamically match the metadata against a lightweight runtime database to determine the best known method corresponding to the current application state.

3 Application Use Cases

We have employed the database components described in Section 2 in two different application contexts.

3.1 Flow in a Driven Cavity

The first parallel application that motivates and validates this work is driven cavity flow [7], which combines lid-driven flow and buoyancy-driven flow in a two-dimensional rectangular cavity. The resulting system of nonlinear PDEs has the form

$$f(u) = 0, \tag{1}$$

where $f : \mathbb{R}^n \to \mathbb{R}^n$. We have selected this model problem because it has properties that are representative of many large-scale, nonlinear PDE-based applications in domains such as computational aerodynamics [2], astrophysics [9], and fusion [27]. We use fully implicit Newton-Krylov methods (see, e.g., [18]) to solve Equation (1) through the two-step sequence of (approximately) solving the Newton correction equation

$$(f'(u^{k-1}))\ \delta u^k = -f(u^{k-1}),$$

in the sense that the linear residual norm $||f'(u^{k-1}))\delta u^k + f(u^{k-1})||$ is sufficiently small and then updating the iterate via $u^k = u^{k-1} + \delta u^k$.

Our intent is to use the performance database components described in Section 2 to implement adaptive linear solver components, such as those described

in [4, 17, 19]. Figure 3 shows a Ccaffeine [1] GUI snapshot of the database and comparator components used to manipulate the performance information for the driven cavity application.

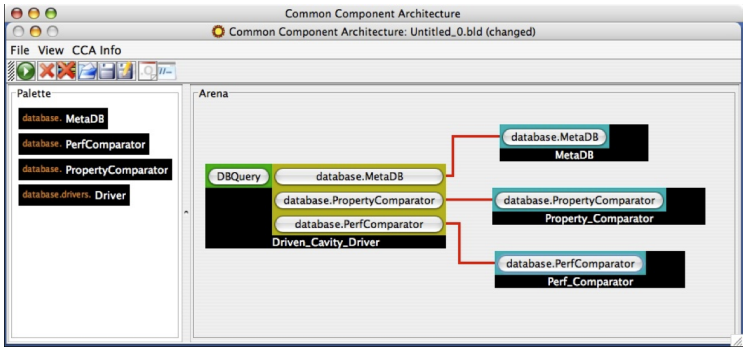

Fig. 3. Database and comparator components used in the driven cavity application.

The performance data is stored in the database by using the `MetDB` component, which provides the *MetaDB* port. Two components provide the *Comparator* port: the `PropertyComparator` and the `PerfComparator`. The `PropertyComparator` component is used to match a particular application instance to one for which we have stored performance information in the database. Currently our matching algorithm uses the matrix metrics computed with Anamod [8]. The `PerfComparator` component analyzes the application-specific performance information to determine the best performing configuration, currently using a least squares approach.

3.2 Combustion

The Computational Facility for Reacting Flow Science project [21] has developed a CCA toolkit for simulating flames on block-structured adaptive meshes. These simulations start with a coarse and uniform grid. The grid is then recursively refined in areas where the accuracy is too low, creating a dynamic grid hierarchy that always conforms to the maximum acceptable error.

For efficient use of the toolkit on parallel computers, the dynamic resource allocation makes it necessary to repeatedly repartition and redistribute the grid hierarchy over the participating processors. The partitioning process must not only take the computations and the CPU performance into account, but also all other factors that contribute to the run-time: communication volume, synchronization delays, data migration between partitions and the performance and utilization of the interconnect. Thus, to minimize the run-time, the current state of the application and the hardware must both be considered. This is non-trivial because the basic conditions for how to allocate hardware resources may change dramatically during run-time.

No single partitioning algorithm is the best choice for all conditions [24]. Instead, good-performing partitioning algorithms need to be dynamically selected and invoked during run-time by a meta-partitioner [12, 13, 25, 26]. The meta-partitioner is a framework that automatically selects, configures, and invokes the best predicted partitioning algorithm with respect to the current application and computer state.

At the core of the meta-partitioner is a compilation of performance data for a large number of partitioning algorithm and SAMR application states. Each of the application states has been carefully characterized using metrics like amount of refined area, aspect ratio, density of patches, and patch sizes. During run-time, the current application state is compared against the characteristics of the application states used to collect the the performance data. The most similar stored application state is recorded. Given the most similar application state, the partitioning algorithm with the best performance is selected and invoked. The workflow of the meta-partitioner is illustrated in Figure 4.

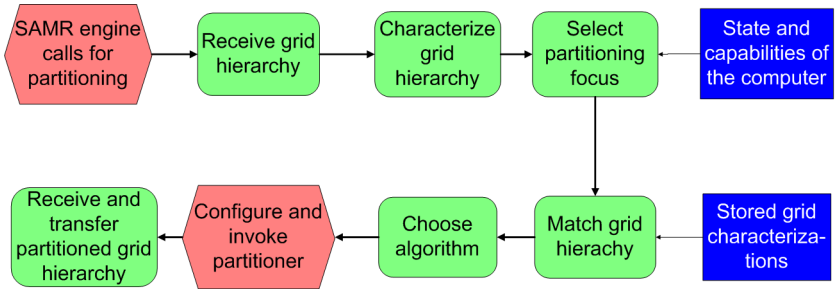

Fig. 4. Sample scenario showing the meta-partitioner role in the workflow of an application.

Without an accurate matching of the current and stored application states, the stored partitioning performance data will not be valid for the current application state. This can potentially result in the selection of bad-performing partitioning algorithms.

The matching of the current and the stored application state is performed in the newly developed `CharacterizationComparator` component. The characteristic data for all application states are stored in the database by using the `MetaDB` component. At each re-partitioning, the characteristics of the current application state are sent to the `CharacterizationComparator`. A weighted least square sum is computed for each combination of current and stored application characteristics. Accessing and using the MetaDB during run-time has several benefits. The implementation of the matching is can be made less involved. It is also easy to expand the MetaDB with new application states. The accuracy of the matching is heavily influenced by the weights used for the least square

sum. Using the MetaDB, these weights can be easily tuned for the most accurate matching.

4 Related Work

Adaptive software for scientific computing is an area of emerging research, as evidenced by numerous recent projects and related work [5,8,10,14–16,23,28,30–33]. Many approaches to addressing different aspects of adaptive execution are represented in these projects, from compiler-based techniques to development of new numerical adaptive algorithms.

Unlike these efforts, our approach is specifically targeted at large-scale parallel computations and relies on high-level interface specifications and technologies tailored for scientific computing. In designing our CQoS interfaces and middleware components, we rely on the existing high-performance infrastructure provided by the CCA, in which multiple component implementations conforming to the same external interface standard are interoperable, and the runtime system ensures that the overhead of component substitution is negligible.

A substantial number of tools for performance analysis exist, including TAU and PerfExplorer [11], Prophesy [29], SvPablo [22], and others. Each tool defines its own performance data representation and storage, from custom ASCII representations or XML to SQL, DB2, or Oracle databases. To our knowledge, the research discussed in this paper is the first attempt to provide language-independent component interfaces and corresponding implementations for performance database manipulation, specifically targeting parallel scientific applications. This approach supports multiple underlying representations and does not preclude the use of existing non-component performance analysis tools.

5 Conclusions and Future Work

We have introduced our prototype component software for managing performance data and associated metadata for high-performance component applications. These components are part of larger CQoS infrastructure, which has the goal of enabling automated component selection and (re)configuration of component-based scientific codes.

This paper focused mainly on the components that provide support for storing and manipulating performance information and associated metadata. There is also related ongoing work on the analysis infrastructure for identifying performance bottlenecks and deriving performance metrics that can be used later to identify configuration parameters and component instances that satisfy given performance or quality requirements.

Acknowledgments

We thank all members of the CCA Forum for stimulating discussions on high-performance scientific software. This work was supported by the Mathematical,

Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Dept. of Energy, under Contract DE-AC02-06CH11357.

References

1. Allan, B., Armstrong, R., Lefantzi, S., Ray, J., Walsh, E., Wolfe, P.: Ccaffeine – a CCA component framework for parallel computing. `http://www.cca-forum.org/ccafe/` (2003)
2. Anderson, W.K., Gropp, W.D., Kaushik, D.K., et al: Achieving high sustained performance in an unstructured mesh CFD application. In: SC99 (1999)
3. Bernholdt, D.E., Allan, B.A., Armstrong, R., Bertrand, F., Chiu, K., Dahlgren, T.L., Damevski, K., Elwasif, W.R., Epperly, T.G.W., Govindaraju, M., Katz, D.S., Kohl, J.A., Krishnan, M., Kumfert, G., Larson, J.W., Lefantzi, S., Lewis, M.J., Malony, A.D., McInnes, L.C., Nieplocha, J., Norris, B., Parker, S.G., Ray, J., Shende, S., Windus, T.L., Zhou, S.: A component architecture for high-performance scientific computing. International Journal of High-Performance Computing Applications pp. 215–229 (2006)
4. Bhowmick, S., Kaushik, D., McInnes, L., Norris, B., Raghavan, P.: Parallel adaptive solvers in compressible PETSc-FUN3D simulations. In: Proceedings of the 17th International Conference on Parallel CFD, Aug 2005 (2005)
5. Bramley, R., Gannon, D., Stuckey, T., Villacis, J., Balasubramanian, J., Akman, E., Berg, F., Diwan, S., Govindaraju, M.: The Linear System Analyzer. In: Enabling Technologies for Computational Science. Kluwer (2000)
6. CCA Forum: CCA Forum homepage. `http://www.cca-forum.org/` (2008)
7. Coffey, T., Kelley, C., Keyes, D.: Pseudo-transient continuation and differential algebraic equations. SIAM J. Sci. Comp 25, 553–569 (2003)
8. Dongarra, J., Eijkhout, V.: Self-adapting numerical software for next generation applications. International Journal of High Performance Computing Applications 17, 125–131 (2003), also LAPACK Working Note 157, ICL-UT-02-07
9. Fryxell, B., Olson, K., Ricker, P., et al: FLASH: An adaptive-mesh hydrodynamics code for modeling astrophysical thermonuclear flashes. Astrophys. J. Suppl. (2000)
10. Houstis, E.N., Catlin, A.C., Rice, J.R., Verykios, V.S., Ramakrishnan, N., Houstis, C.E.: A knowledge/database system for managing performance data and recommending scientific software. ACM Transactions on Mathematical Software 26(2), 227–253 (2000)
11. Huck, K.A., Malony, A.D., Shende, S., Morris, A.: Scalable, automated performance analysis with tau and perfexplorer. In: Parallel Computing (ParCo). Aachen, Germany (2007)
12. Johansson, H.: Performance Characterization and Evaluation of Parallel PDE Solvers. Licentiate, University of Uppsala, Library, Box 510, SE-751, 20 Uppsala, Sweden (Nov 2006)
13. Johansson, H., Steensland, J.: A performance characterization of load balancing algorithms for parallel SAMR applications. Tech. Rep. 2006-047, Department of Information Technology, Uppsala University, Sweden (2006), available at `http://www.it.uu.se/research/reports/2006`
14. Keahey, K., Beckman, P., Ahrens, J.: Ligature: A component architecture for high-performance applications. International Journal of High-Performance Computing Applications (14) (2000)

15. Liu, H., Parashar, M.: Enabling self-management of component based high-performance scientific applications. In: Proceedings of the 14th IEEE International Symposium on High Performance Distributed Computing. IEEE Computer Society Press (July 2005)
16. McCracken, M.O., Snavely, A., Malony, A.: Performance modeling for dynamic algorithm selection. In: Proc. of the International Conference on Computational Science (ICCS'03), LNCS. vol. 2660, pp. 749–758. Springer, Berlin (2003)
17. McInnes, L.C., Norris, B., Bhowmick, S., Raghavan, P.: Adaptive sparse linear solvers for implicit CFD using Newton-Krylov algorithms. In: Proceedings of the Second MIT Conference on Computational Fluid and Solid Mechanics, June 17-20, 2003, Cambridge, MA. vol. 2, pp. 1024–1028. Elsevier (2003)
18. Nocedal, J., Wright, S.J.: Numerical Optimization. Springer-Verlag (1999)
19. Norris, B., McInnes, L., Veljkovic, I.: Computational quality of service in parallel CFD. Argonne National Laboratory preprint ANL/MCS-P1283-0805 (2005), submitted to Proc. of the 17th International Conference on Parallel CFD, Aug 2005
20. Norris, B., Ray, J., Armstrong, R., McInnes, L.C., Bernholdt, D.E., Elwasif, W.R., Malony, A.D., Shende, S.: Computational quality of service for scientific components. In: Proceedings of International Symposium on Component-Based Software Engineering (CBSE7), Edinburgh, Scotland (2004)
21. (PI), H.N.: Computational facility for reacting flow science (CFRFS). `http://www.ca.sandia.gov/cfrfs/` (2008)
22. de Rose, L.A., Reed, D.A.: SvPablo: A multi-language architecture-independent performance analysis system. In: ICPP '99: Proceedings of the 1999 International Conference on Parallel Processing. p. 311. IEEE Computer Society, Washington, DC, USA (1999)
23. Sosonkina, M.: Runtime adaptation of an iterative linear system solution to distributed environments. In: Applied Parallel Computing, PARA'2000. Lecture Notes in Computer Science, vol. 1947, pp. 132–140. Springer-Verlag, Berlin (2001)
24. Steensland, J.: Efficient partitioning of dynamic structured grid hierarchies. Ph.D. thesis, University of Uppsala, Library, Box 510, SE-751, 20 Uppsala, Sweden (2002)
25. Steensland, J., Chandra, S., Parashar, M.: An application-centric characterization of domain-based SFC partitioners for parallel SAMR. IEEE Transactions on Parallel and Distributed Systems 13(12), 1275–1289 (Dec 2002)
26. Steensland, J., Thuné, M., Chandra, S., Parashar, M.: Characterization of domain-based partitioners for parallel SAMR applications. In: Proceedings of the IASTED International Conference Parallel and Distributed Computing and Systems. ACTA Press (2000)
27. Tang, X.Z., Fu, G.Y., Jardin, S.C., et al: Resistive magnetohydrodynamics simulation of fusion plasmas. Tech. Rep. PPPL-3532, Princeton Plasma Physics Laboratory (2001)
28. Tapus, C., Chung, I.H., Hollingsworth, J.K.: Active Harmony: Towards automated performance tuning. In: Proceedings of SC02 (2002)
29. Taylor, V., Wu, X., Stevens, R.: Prophesy: An infrastructure for performance analysis and modeling of parallel and grid applications. SIGMETRICS Perform. Eval. Rev. 30(4), 13–18 (2003)
30. Vetter, J.S., Worley, P.H.: Asserting performance expectations. In: Proceedings of SC02 (2002)
31. Vuduc, R., Demmel, J., Bilmes, J.: Statistical models for empirical search-based performance tuning. International Journal of High Performance Computing Applications 18(1), 65–94 (February 2004)

32. Whaley, R.C., Petitet, A.: Minimizing development and maintenance costs in supporting persistently optimized BLAS. Software: Practice and Experience 35(2), 101–121 (February 2005), http://www.cs.utsa.edu/~whaley/papers/spercw04.ps
33. Zhang, K., Damevski, K., Venkatachalapathy, V., Parker, S.: SCIRun2: A CCA framework for high performance computing. In: Proceedings of the 9th International Workshop on High-Level Parallel Programming Models and Supportive Environments (HIPS 2004). IEEE Press, Santa Fe, NM (April 2004), to appear